

2018-2019

SWING

OGBONE Irédon Arif

Table des matières

Introduction.....	3
Présentation.....	3
I)Présentation des classes.....	4
1. La classe JFrame :.....	4
2. La classe JButton(Bouton) :.....	7
3. La classe JLabel(un libellé ou une étiquette) :.....	8
4. La classe Color (les couleurs) :.....	9
5. Layout manager (Gestionnaires de présentation).....	10
a)BorderLayout :.....	10
b)FlowLayout :.....	11
c)GridLayout.....	12
6. La classe JPanel (Panneau).....	14
7. Les classes JMenuBar, JMenu, JMenuItem.....	15
8. La classe AbstractButton.....	16
9. La classe JTextField (champ de texte).....	16
10. La classe JTextArea.....	17
11. La classe JPasswordField.....	19
12. La classe ButtonGroup.....	20
13. La classe JbuttonRadio (bouton radio).....	20
14. La classe JCheckBox.....	21
15. La classe JComboBox.....	22
16. La classe JTable.....	23
17. La classe JOptionPane.....	24
II)Les événements.....	26
1. Interagir avec le bouton : L'interface ActionListener.....	26
2. Interagir avec la souris : l'interface MouseListener.....	28
3. Contrôle du clavier : l'interface KeyListener.....	29
4. L'interface WindowListener.....	30

Introduction

Swing est une bibliothèque java permettant de créer des interfaces utilisateurs graphiques (Graphical User Interface(GUI)). Swing peut-être vu comme une version améliorée du paquetage AWT (Abstractt Window Toolkit) qui est un ancien paquetage destiné pour concevoir des interfaces de fenêtre. Mais d'autres classes de AWT sont nécessaires dans l'utilisation de Swing. Swing fournit les classes pour la représentation des différents éléments d'interfaces graphiques : fenêtre, bouton, menus, liste déroulante etc., et la gestion des événements. Les classes de Swing se trouvent dans le paquetage **javax.swing**.

Dans cette présentation, nous utiliserons que les composants swing ; mais nous utiliserons les objets issus du package awt pour interagir et communiquer avec les composants swing.

Présentation

Les principaux éléments graphiques de Swing sont des **composants**. Ils dérivent de la classe **javax.swing.JComponent**. Pour être utilisé, un composant doit le plus souvent être placé dans un **conteneur** (java.awt.Container). Les objets conteneurs regroupent les composants, les organisent pour les afficher grâce à un gestionnaire de placement ou de présentation.

Les fonctionnalités des composants se décomposent essentiellement en deux catégories : celles responsables de l'apparence et celles chargées du comportement.

Pour ce qui est de l'apparence, il s'agit par exemple de la visibilité, la taille, la couleur, la police, etc.

Pour le comportement, il s'agit de la réaction du composant aux événements contrôlés par l'utilisateur. Les événements sont des actions qui peuvent être un clic de souris, le déplacement de la souris, le redimensionnement d'un objet, son masquage, sa sélection, la frappe d'une touche du clavier, etc. Pour ces actions, Swing (pour simplifier) envoie une notification d'événement aux objets qui se sont abonnés en tant que "listener" auprès du composant qui est à l'origine de l'événement. Ce listener doit posséder les méthodes de réaction à l'événement et celles-ci sont alors invoquées. C'est le gestionnaire d'événements de Swing qui est en charge de ce mécanisme. Le mécanisme est similaire à celui d'autres langages, comme javascript par exemple.

Un conteneur est un composant qui contient d'autres composants et qui les gouvernent. Les conteneurs les plus utilisés sont JFrame et JPanel. Il faut noter que les Panels sont aussi des composants. En effet un conteneur peut lui-même contenir d'autres conteneurs et on crée ainsi une hiérarchie d'imbrications de composants. La méthode add des conteneurs permet de leur ajouter de nouveaux composants. Il est rare qu'un composant autre qu'un conteneur ait une existence en dehors d'un conteneur.

I) Présentation des classes

1. La classe JFrame :

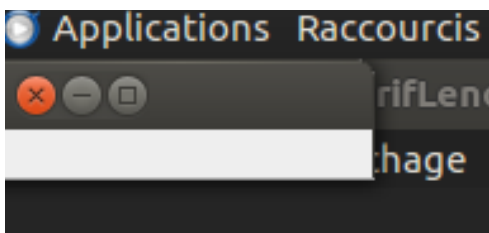
Un objet JFrame est une fenêtre de base qui inclue une bordure et les trois boutons usuels de réduction, d'agrandissage et de fermeture de la fenêtre.
La taille de la fenêtre JFrame est définie en utilisant la méthode setSize.

```
myFrame.setSize(WIDTH, HEIGHT);
```

Les paramètres WIDTH et HEIGHT sont de type int et l'unité de mesure est en pixels.

Présentation 1.1.1 Première démonstration de JFrame

```
1 import javax.swing.JFrame;
2
3 public class DemoFrame{
4     public static void main(String[] args){
5         JFrame fenetre = new JFrame();
6         fenetre.setVisible(true);
7     }
8 }
```

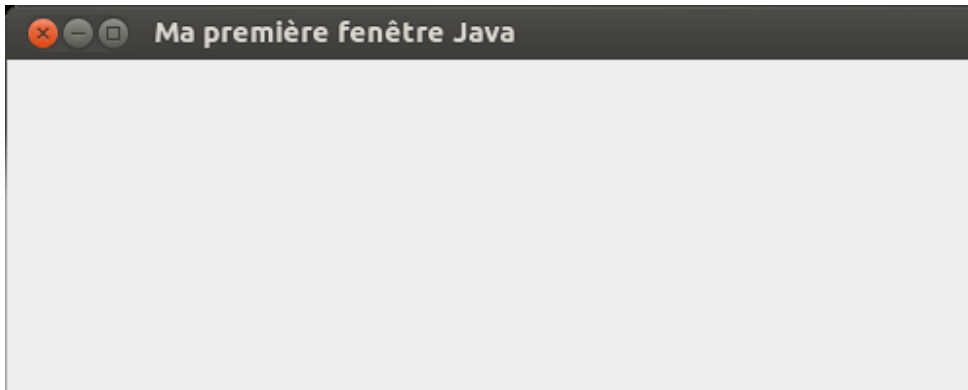


Vous remarquez que notre fenêtre est très petite et cachée dans le coin supérieur gauche de l'écran. Pour que notre fenêtre soit parfaite, Il faudra donc que nous précisons sa taille, son titre puis sa position sur l'écran (au centre serait parfait).

Présentation 1.1.2 Deuxième démonstration de JFrame

```
1 import javax.swing.JFrame;
2 public class Demo2Frame{
3     public static void main(String[] args){
4         JFrame fenetre = new JFrame();
5         //Définit un titre pour notre fenêtre
6         fenetre.setTitle("Ma première fenêtre Java");
7         //Définit la taille de notre fenêtre
8         fenetre.setSize(400, 200);
9         //Nous demandons maintenant à notre objet de se
10        positionner au centre
11        fenetre.setLocationRelativeTo(null);
12        //Termine le processus lorsqu'on clique sur la croix
13        rouge de la fenêtre
14        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
15        //rendre visible la fenêtre
16        fenetre.setVisible(true);
17    }
18 }
```

```
15     }  
16 }
```



Présentation 1.1.3 Troisième démonstration de JFrame

```
1 import javax.swing.JFrame;  
2  
3 public class Demo3Frame extends JFrame{  
4     public Demo3Frame(){  
5         this.setTitle("Ma première fenêtre Java");  
6         this.setSize(400, 200);  
7         this.setLocationRelativeTo(null);  
8         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
9     }  
10    public static void main(String[] args) {  
11        Demo3Frame frame3 = new Demo3Frame();  
12        frame3.setVisible(true);  
13    }  
14 }
```

Ce code donne le même résultat que le code précédent.

Quelques fonctions de la classe de la classe **Jframe** :

Fonctions	Rôles
public JFrame()	Constructeur qui crée un objet de la classe JFrame.
public JFrame(String title)	Constructeur qui crée un objet de la classe JFrame avec le titre donné comme un argument.
public void setDefaultCloseOperation(int operation)	Indique l'action qui se réalisera par défaut si l'utilisateur clique le bouton de fermeture de la fenêtre.
public void setSize(int width, int height)	Définit la taille de la fenêtre appelante pour qu'elle ait une largeur et une hauteur précisée.
public void setTitle(String title)	Indique le titre de la fenêtre. Le titre est

	passé en argument de la fonction.
Public void setLocation(int x, int y)	Spécifie où doit se situer la fenêtre sur l'écran. Les coordonnées sont exprimées en pixels et sont basées sur un repère dont l'origine est représentée par le coin supérieur gauche.
Public void setResizable(boolean argument)	false empêche le redimensionnement de la fenêtre tandis que true l'autorise.
public void add(Component componentAdded)	Ajoute un composant à la fenêtre.
public void setLayout(LayoutManager manager)	Définit le gestionnaire de présentation.
public void setJMenuBar(JMenuBar menubar)	Définit la barre de menu de la fonction appelante
public void dispose()	Élimine la fenêtre appelante et ses sous-composants.
Public void setVisible(boolean argument)	permet à la fenêtre JFrame d'être visible à l'écran. Elle prend un paramètre de type booléen. La fenêtre est visible lorsque l'argument est true.

Les arguments que peut prendre la fonction **setDefaultCloseOperation** sont :

DO_NOTHING_ON_CLOSE : La fenêtre ne fait rien, mais s'il y a un window listener, la fonction correspondante est appelée. (les windows listener sont expliqués dans la partie II).

HIDE_ON_CLOSE : Ferme la fenêtre mais le programme ne se termine pas.

DISPOSE_ON_CLOSE : Ferme la fenêtre mais le programmation reste actif pendant un petit temps avant de se terminer.

EXIT_ON_CLOSE : Permet de quitter la fenêtre.

Si vous ne spécifier pas d'action, HIDE_ON_CLOSE est l'action prise par défaut.

TD1

Créer une classe MaFenetre qui :

- hérite de JFrame,
- a pour titre *Fenêtre d'expérimentation*,
- est située au centre,
- a pour hauteur 250pixels et largeur 400 pixels
- ne se ferme pas lorsqu'on clique sur la croix rouge,
- ne peut pas être redimensionner.

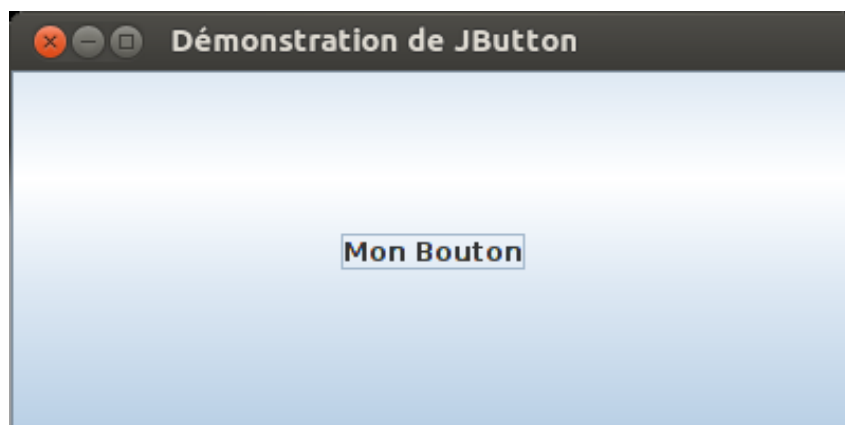
2. La classe JButton(Bouton) :

Un objet de la classe JButton se comporte comme un bouton, un clique avec la souris permet d'appuyer sur le bouton. Lorsqu'on crée un objet de la classe JButton en utilisant new, une chaîne peut être donnée au constructeur comme argument et cette chaîne sera visible sur le bouton.

L'objet JButton est ajouté à un conteneur (objet JFrame ou JPanel) en utilisant la méthode add, avec le conteneur comme l'objet appelant et le JButton comme argument.

Présentation 1.2 Démonstration de JButton

```
1 import javax.swing.JFrame;
2 import javax.swing.JButton;
3
4 public class DemoBouton extends JFrame{
5     public DemoBouton(){
6         this.setTitle("Ma première fenêtre Java");
7         this.setSize(400, 200);
8         this.setLocationRelativeTo(null);
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        //Création du bouton
11        JButton bouton = new JButton("Mon Bouton");
12        //Ajout du bouton dans la fenêtre
13        add(bouton);
14
15    }
16    public static void main(String[] args) {
17        DemoBouton demobouton = new DemoBouton();
18        demobouton.setVisible(true);
19    }
20 }
```



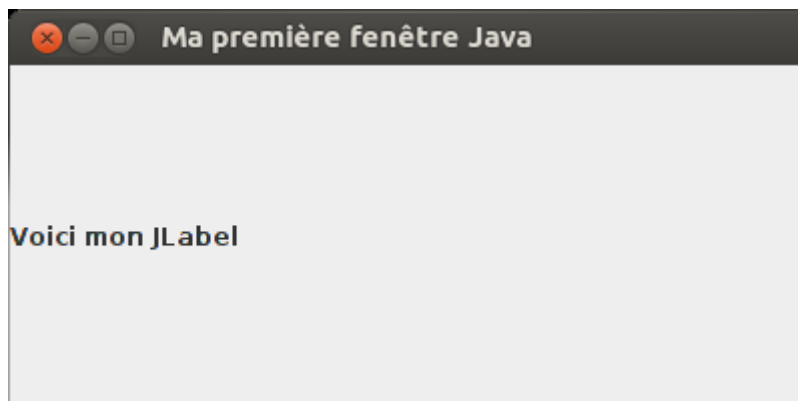
3. La classe JLabel(un libellé ou une étiquette) :

Un label est un objet de la classe JLabel issu du paquetage javax.swing. Cet objet se comporte comme un libellé. Il permet donc d'afficher du texte. Le texte du label est donné comme argument au constructeur du JLabel.

Le label peut être aussi ajouté à l'objet JFrame grâce à la méthode add.

Présentation 1.3 Démonstration de JLabel

```
1 import javax.swing.JFrame;
2 import javax.swing.JLabel;
3
4 public class DemoLabel extends JFrame{
5     public DemoLabel(){
6         this.setTitle("Ma première fenêtre Java");
7         this.setSize(400, 200);
8         this.setLocationRelativeTo(null);
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        //Création du label
11        JLabel label = new JLabel("Voici mon JLabel");
12        //Ajout du label dans la fenêtre
13        add(label);
14    }
15    public static void main(String[] args) {
16        DemoLabel label = new DemoLabel();
17        label.setVisible(true);
18    }
19 }
```



4. La classe Color (les couleurs) :

Vous pouvez définir la couleur d'une fenêtre ou tout autre objet GUI.

Une couleur est une instance de la classe Color. Cette dernière se trouve dans le paquetage java.awt (java.awt.Color).

La méthode permettant de spécifier la couleur d'un objet est setBackground. Elle prend en argument la couleur.

Les noms des couleurs sont en anglais et l'orthographe doit être respecté. Ces noms peuvent être écrits en lettres majuscules ou minuscules.

les couleurs disponibles :

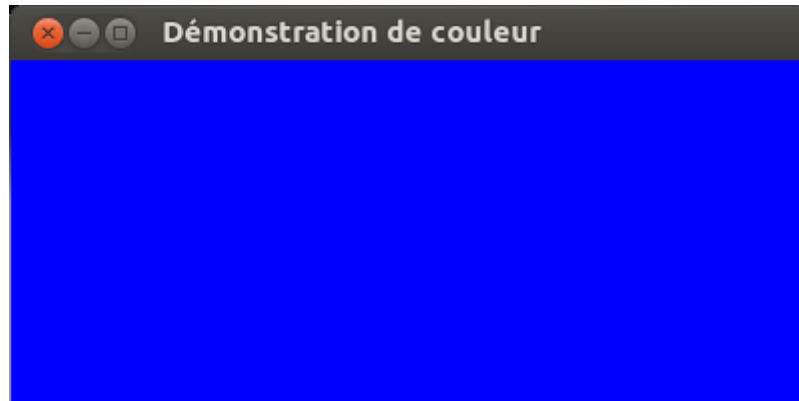
Color.BLACK
Color.BLUE
Color.CYAN
Color.DARK_GRAY
Color.GRAY
Color.GREEN
Color.LIGHT_GRAY
Color.MAGENTA
Color.ORANGE
Color.PINK
Color.RED
Color.WHITE
Color.YELLOW

Par exemple, colorions en bleu notre fenêtre.

NB : Pour colorier toute une fenêtre, il faudra récupérer d'abord son contenu(intérieur) avec la méthode getContentPane() et ensuite colorier le contenu.

Présentation 1.4.1 Présentation de Color

```
1 import javax.swing.JFrame;  
2 import java.awt.*;  
3  
4 public class ColorFrame{  
5     public static void main(String[] args){  
6         JFrame fenetre = new JFrame();  
7         fenetre.setTitle("Démonstration de couleur");  
8         fenetre.setSize(400, 200);  
9         fenetre.setLocationRelativeTo(null);  
10        fenetre.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
11        //Colorie l'intérieur ou le fond de la fenêtre en bleu  
12        fenetre.getContentPane().setBackground(Color.blue);  
13        fenetre.setVisible(true);  
14    }  
15 }
```

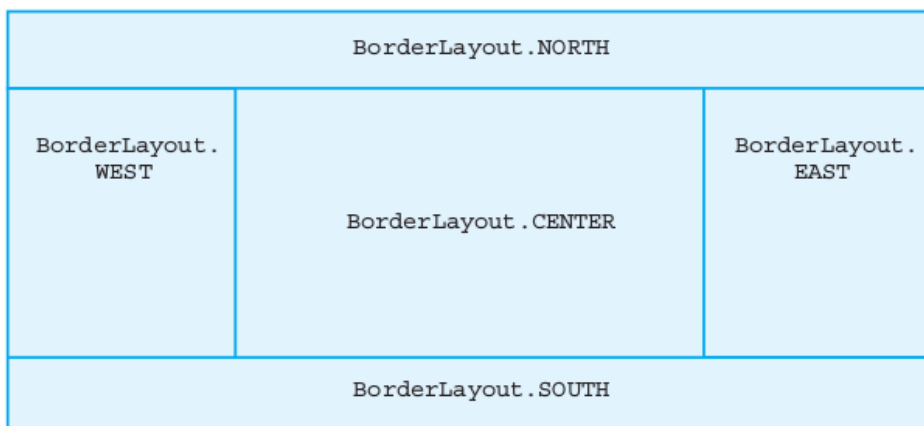


5. Layout manager (Gestionnaires de présentation)

Les gestionnaires de présentation permettent de décrire comment les composants seront disposés ou placés sur la fenêtre. Ils sont dans le paquetage `java.awt`. `setLayout` est la méthode permettant de spécifier le gestionnaire de présentation d'un conteneur (objet `JFrame` ou `JPanel`). Elle prend en paramètre une instance du gestionnaire de présentation.

a) BorderLayout :

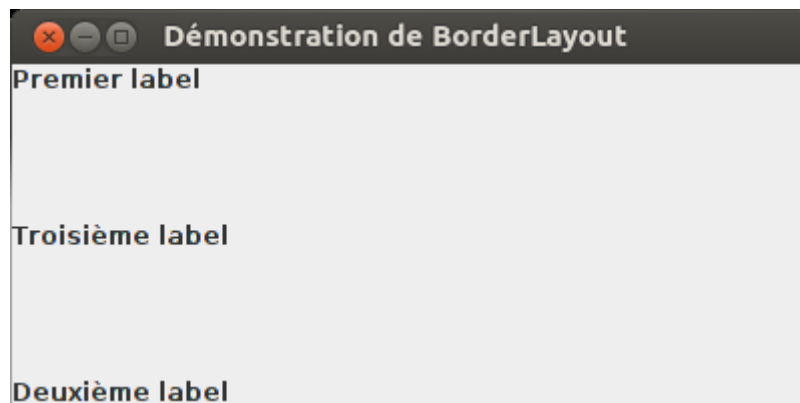
Dispose les composants dans 5 régions : Nord, Sud, Ouest, Est et Centre.



Présentation 1.5.1 Présentation de BorderLayout

```
1 import javax.swing.JFrame;
2 import javax.swing.JLabel;
3 import java.awt.BorderLayout;
4
5 public class BorderDemo extends JFrame{
6     public BorderDemo(){
7         this.setTitle("Démonstration de BorderLayout");
8         this.setSize(400, 200);
9         this.setLocationRelativeTo(null);
10        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
11         this.setLayout(new BorderLayout());
12         //Création du label1
13         JLabel label1 = new JLabel("Premier label");
14         //ajoute le label1 dans la fenêtre puis le positionne au
    nord
15         add(label1, BorderLayout.NORTH);
16         JLabel label2 = new JLabel("Deuxième label");
17         //ajoute le label2 dans la fenêtre puis le positionne au
    sud
18         add(label2, BorderLayout.SOUTH);
19         JLabel label3 = new JLabel("Troisième label");
20         //ajoute le label3 dans la fenêtre puis le positionne au
    centre
21         add(label3, BorderLayout.CENTER);
22     }
23     public static void main(String[] args) {
24         BorderDemo label = new BorderDemo();
25         label.setVisible(true);
26     }
27 }
```



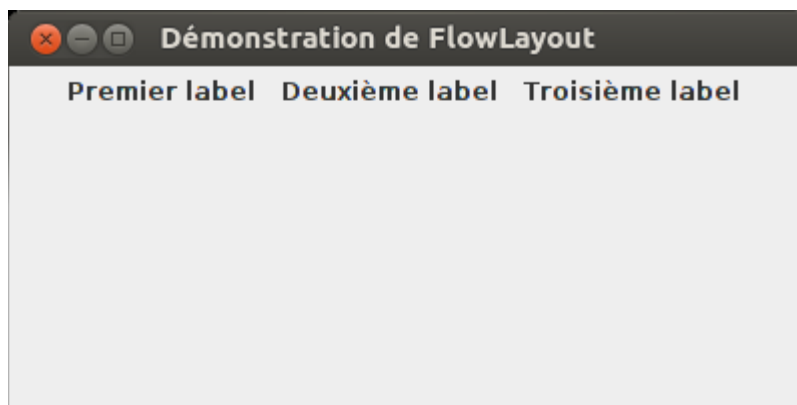
b) FlowLayout :

Dispose les composants de la gauche vers la droite dans l'ordre où ils sont écrits.

Présentation 1.5.2 Démonstration de FlowLayout

```
1 import javax.swing.JFrame;
2 import javax.swing.JLabel;
3 import java.awt.FlowLayout;
4
5 public class FlowDemo extends JFrame{
6     public FlowDemo(){
7         setTitle("Démonstration de FlowLayout");
8         setSize(400, 200);
9         setLocationRelativeTo(null);
10        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
11         this.setLayout(new FlowLayout());
12         //Création du label1
13         JLabel label1 = new JLabel("Premier label ");
14         //ajoute le label1 dans la fenêtre
15         add(label1);
16         JLabel label2 = new JLabel(" Deuxième label  ");
17         //ajoute le label2 dans la fenêtre
18         add(label2);
19         JLabel label3 = new JLabel(" Troisième label ");
20         //ajoute le label3 dans la fenêtre
21         add(label3);
22     }
23     public static void main(String[] args) {
24         FlowDemo flow = new FlowDemo();
25         flow.setVisible(true);
26     }
27 }
```



c) GridLayout

Dispose les composants dans une grille à deux dimensions avec un nombre de lignes et de colonnes.

Exemple : `unConteneur.setLayout(new GridLayout(2, 3))`

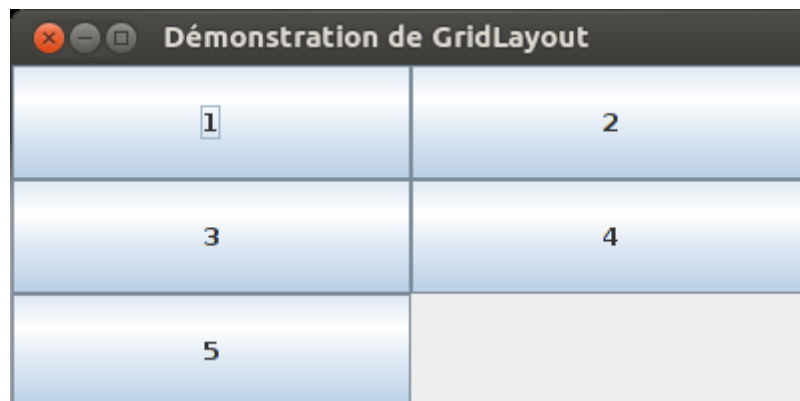
Les deux nombres mis en paramètre spécifient respectivement le nombre de lignes et de colonnes. Les éléments sont disposés à partir de la case située en haut à gauche.

Les méthodes **setHgap** et **setVgap** permettent respectivement d'ajouter de l'espace entre les colonnes et les lignes (H comme horizontal et V comme vertical).

Présentation 1.5.3 Démonstration GridLayout

```
1 import java.awt.GridLayout;
2 import javax.swing.JButton;
3 import javax.swing.JFrame;
4
5 public class GridDemo extends JFrame{
6     public GridDemo() {
```

```
7         this.setTitle("Démonstration de GridLayout");
8         this.setSize(400, 200);
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        this.setLocationRelativeTo(null);
11        //On définit le layout à utiliser sur le content pane
12        //Trois lignes sur deux colonnes
13        this.setLayout(new GridLayout(3, 2));
14        //On ajoute les boutons dans la JFrame
15        add(new JButton("1"));
16        add(new JButton("2"));
17        add(new JButton("3"));
18        add(new JButton("4"));
19        add(new JButton("5"));
20        setVisible(true);
21    }
22    public static void main(String[] args) {
23        GridDemo grid = new GridDemo();
24        grid.setVisible(true);
25    }
26 }
```



TD2

1. Créer une fenêtre puis définissez un gestionnaire de placement et ajouter des éléments, pour qu'on ait la présentation suivante :

Premier label

Deuxième label

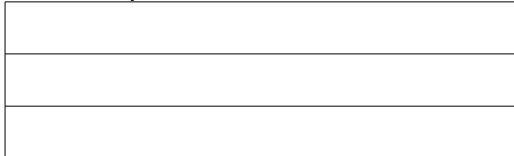
Troisième label

2. Supposons que vous aviez créé une fenêtre dérivé de JFrame et supposons que vous voulez spécifier un gestionnaire de présentation pour avoir la présentation suivante (une ligne, avec trois colonnes) :



Quel sera l'argument de `setLayout` ?

3. Supposons que la situation est la même que la précédente ; sauf que vous voulez la présentation suivante :



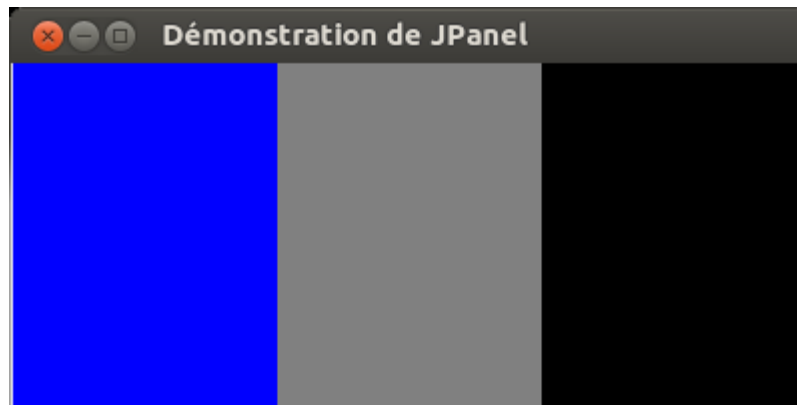
Quel sera l'argument de `setLayout` ?

6. La classe `JPanel` (Panneau)

Un panel est un objet de la classe `JPanel`. Un panel est un conteneur donc peut contenir d'autres composants swing tels que les boutons, les labels, d'autres panels... L'une des fonctions principales d'un objet `JPanel` est de subdiviser un `JFrame`(ou tout autre conteneur) en plusieurs parties.

Présentation 1.5.1 Démonstration `JPanel`

```
1 import javax.swing.JFrame;
2 import javax.swing.JPanel;
3 import java.awt.Color;
4 import java.awt.GridLayout;
5
6 public class PanelDemo extends JFrame{
7     public PanelDemo (){
8         this.setTitle("Démonstration de JPanel");
9         this.setSize(400, 200);
10        this.setLocationRelativeTo(null);
11        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        this.setLayout(new GridLayout(1, 3));
13        JPanel panelBleu = new JPanel();
14        panelBleu.setBackground(Color.blue);
15        this.add(panelBleu);
16        JPanel panelGris = new JPanel();
17        panelGris.setBackground(Color.gray);
18        this.add(panelGris);
19        JPanel panelNoire = new JPanel();
20        panelNoire.setBackground(Color.black);
21        this.add(panelNoire);
22    }
23    public static void main(String[] args) {
24        PanelDemo paneldemo = new PanelDemo();
25        paneldemo.setVisible(true);
26    }
27 }
```

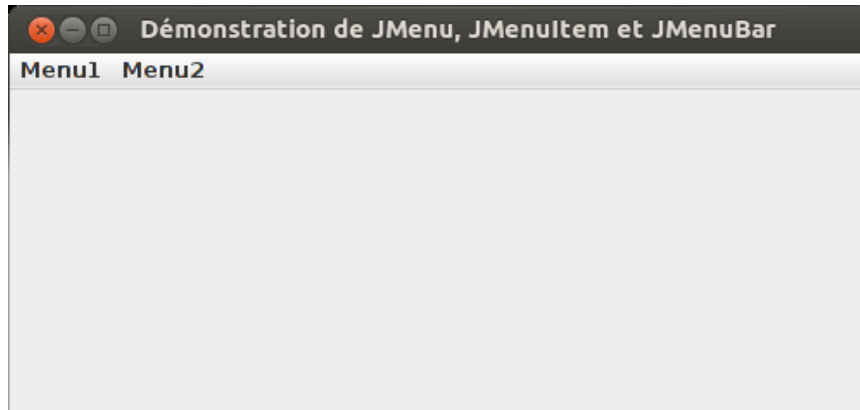


7. Les classes JMenuBar, JMenu, JMenuItem

Un Menu est un objet de la classe JMenu. Une option sur un menu est appelé élément du menu (menu item) qui est un objet de la classe JMenuItem. Les JMenuItem sont placés dans les JMenus et les JMenus sont ensuite placés dans un JMenuBar. Vous pouvez placer autant de JMenuItem que vous voulez dans un menu ; le menu listera ces JMenuItem dans l'ordre dans lequel ils ont été placé.

Présentation 1.7 Démonstration JMenuBar, JMenu, JMenuItem

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class MenuDemo extends JFrame {
5
6     public MenuDemo() {
7         setTitle("Démonstration de JMenu, JMenuItem et
JMenuBar");
8         setSize(400, 200);
9         setLocationRelativeTo(null);
10        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11        JMenu labelMenu = new JMenu("Menu1");
12        JMenu labelMenu2 = new JMenu("Menu2");
13        JMenuItem menuItem1 = new JMenuItem("MenuItem 1");
14        labelMenu.add(menuItem1);
15        JMenuItem menuItem2 = new JMenuItem("MenuItem 2");
16        labelMenu.add(menuItem2);
17        JMenuBar barMenu = new JMenuBar();
18        barMenu.add(labelMenu);
19        barMenu.add(labelMenu2);
20        setJMenuBar(barMenu);
21    }
22    public static void main(String[] args) {
23        MenuDemo menus = new MenuDemo();
24        menus.setVisible(true);
25    }
26 }
```



8. La classe **AbstractButton**.

Les classes `JButton` et `JMenuItem` sont des classes dérivées de la classe abstraite `AbstractButton`.

Toutes les méthodes et propriétés de bases des classes `JButton` et `JMenuItem` sont hérités de la classe `AbstractButton`. C'est la raison pour laquelle ces deux méthodes sont similaires.

Quelques fonctions de la classe **`AbstractButton`**.

Fonctions	Rôles
<code>public void setBackground(Color theColor)</code>	Définit la couleur de fond du composant.
<code>public void addActionListener(ActionListener listener)</code>	Ajoute un <code>ActionListener</code> .
<code>public void removeActionListener(ActionListener listener)</code>	Retirer ou supprimer un <code>ActionListener</code> .
<code>public void setText(String text)</code>	Fait de <code>text</code> le seul texte sur le composant.
<code>public String getText()</code>	Retourne le texte écrit sur le composant, tel que le texte sur un bouton ou la chaîne sur un élément du menu.
<code>public void setPreferredSize(Dimension preferredSize)</code>	Définit la taille préférée d'un bouton.
<code>public void setPreferredSize(new Dimension(int width, int height))</code>	

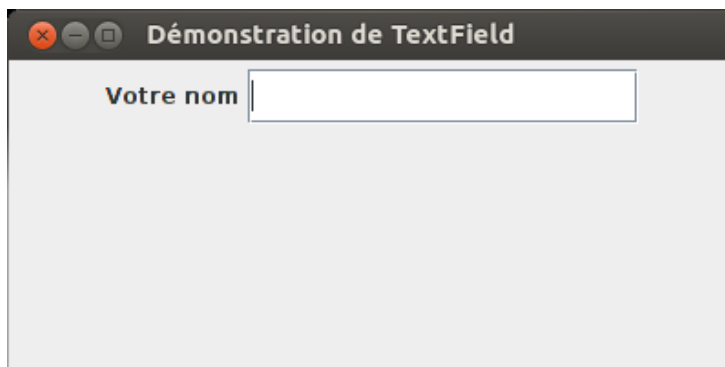
9. La classe JTextField (champ de texte)

Un champ de texte est un objet de la classe JTextField et présenté comme un champ qui permet à l'utilisateur d'entrer une ligne de texte. Son constructeur prend en paramètre un entier qui spécifie le nombre de caractères qui seront au moins visibles dans le champs.

Par exemple définissons un champs dans lequel l'utilisateur devra saisir son nom :

Présentation 1.9 Démonstration de JTexField

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 public class TextFieldDemo extends JFrame {
5     JTextField champNom = new JTextField(15);
6     JLabel label = new JLabel("Votre nom");
7     JPanel panel = new JPanel();
8
9     public TextFieldDemo() {
10         setTitle("Démonstration de TextField");
11         setSize(400, 200);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         setLocationRelativeTo(null);
14         champNom.setPreferredSize(new Dimension(40, 30));
15         Font police = new Font("Arial", Font.BOLD, 14);
16         champNom.setFont(police);
17         panel.add(label);
18         panel.add(champNom);
19         add(panel);
20     }
21     public static void main(String[] args) {
22         TextFieldDemo field = new TextFieldDemo();
23         field.setVisible(true);
24     }
25 }
```

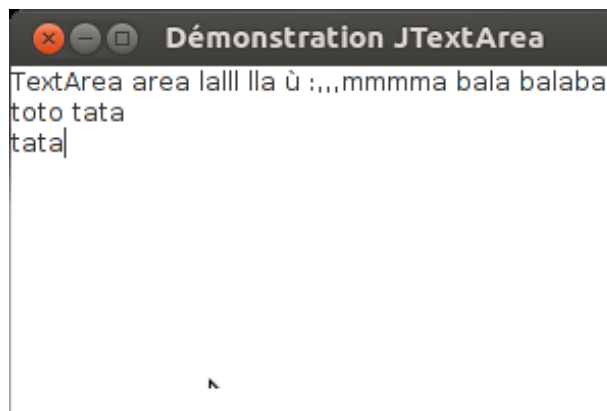


10. La classe JTextArea

Un text Area est un objet de la classe JTextArea. Un text area est similaire à un text field sauf qu'il permet la saisie de plusieurs lignes.

Présentation 1.10.1 Première Démonstration de JTextArea

```
1 import javax.swing.JFrame;
2 import javax.swing.JTextArea;
3
4 public class TextAreaDemo extends JFrame {
5     JTextArea textPane = new JTextArea();
6     public TextAreaDemo() {
7         this.setLocationRelativeTo(null);
8         this.setTitle("Démonstration JTextArea");
9         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10        this.setSize(200, 200);
11        add(textPane);
12        this.setVisible(true);
13    }
14    public static void main(String[] args) {
15        TextAreaDemo fen = new TextAreaDemo();
16    }
17 }
```



Vous remarquez que si vous écrivez trop long jusqu'à dépasser la taille de la fenêtre, le texte devient invisible mais il est bien écrit. Pour résoudre ce problème, nous allons utiliser ce qu'on appelle des « scrolls ». Ce sont de petits ascenseurs positionnés sur le côté et / ou sur le bas de votre fenêtre et qui vous permettent de dépasser les limites de la fenêtre (figure ci-dessous).

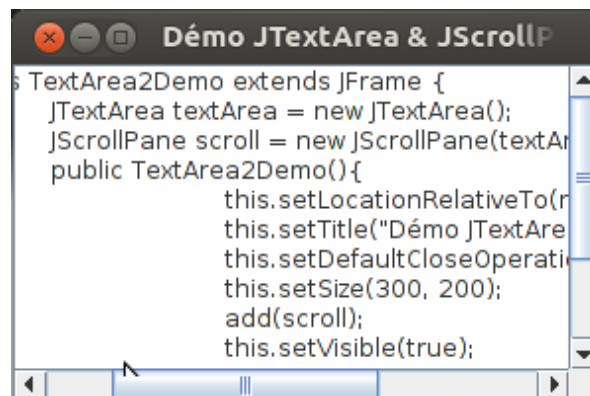
Présentation 1.10.2 Deuxième Démonstration de JTextArea

```
1 import javax.swing.JFrame;
2 import javax.swing.JTextArea;
3 import javax.swing.JScrollPane;
4
5 public class TextArea2Demo extends JFrame {
6     JTextArea textArea = new JTextArea();
```

```

7      JScrollPane scroll = new JScrollPane(textArea);
8      public TextArea2Demo() {
9          this.setLocationRelativeTo(null);
10         this.setTitle("D mo JTextArea & JScrollPane");
11         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12         this.setSize(300, 200);
13         add(scroll);
14         this.setVisible(true);
15     }
16     public static void main(String[] args) {
17         TextArea2Demo fen = new TextArea2Demo();
18     }
19 }

```



Quelques m thodes de la classe **JTextcomponent** :

Fonctions	R�les
public String getText()	Retourne le texte qui est pr�sent� par le composant de texte.
public boolean isEditable()	Retourne vrai si l'utilisateur peut �crire dans le composant de texte. Retourne faux si l'utilisateur n'est pas autoris� � �crire dans le composant de texte.
public void setBackground(Color theColor)	D�finit la couleur de fond du composant de texte.
public void setEditable(boolean argument)	Si l'argument est vrai donc l'utilisateur est autoris� � �crire dans le composant de texte ; dans le cas o� c'est faux, l'utilisateur n'est pas autoris�.
public void setText(String text)	D�finit le texte qui sera pr�sent� par le composant de texte.

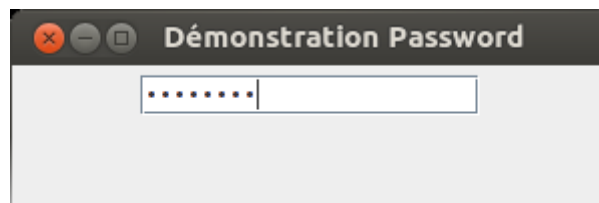
11. La classe JPasswordField

Cette classe permet la saisie d'un texte dont tous les caractères saisis seront affichés sous forme d'un caractères particulier('*', '.', ...).

La méthode **setEchoChar(char)** permet de préciser le caractère qui sera montré lors de la saisie. Par défaut c'est le caractère point (.).

Présentation 1.11 Démonstration de JPasswordField

```
1 import java.awt.Dimension;
2 import javax.swing.*;
3
4 public class DemoPassword {
5     public static void main(String argv[]) {
6         JFrame fenetre = new JFrame("Démonstration Password");
7         fenetre.setSize(300, 100);
8         fenetre.setLocationRelativeTo(null);
9         JPanel pannel = new JPanel();
10        JPasswordField passwordField1 = new JPasswordField (15);
11        passwordField1.setPreferredSize(new Dimension(100,20 ));
12        pannel.add(passwordField1);
13        fenetre.getContentPane().add(pannel);
14        fenetre.setVisible(true);
15    }
16 }
```



12. La classe ButtonGroup

Cette classe permet de gérer un ensemble de boutons en garantissant qu'un seul bouton du groupe sera sélectionné.

Pour ajouter un bouton au groupe on utilise la méthode **add** en passant le bouton en argument.

13. La classe JRadioButton (bouton radio)

Un bouton radio est un objet de la classe **JRadioButton**. A un instant donné, un seul des boutons radios associé à un même groupe de boutons (**ButtonGroup**) peut être sélectionné.

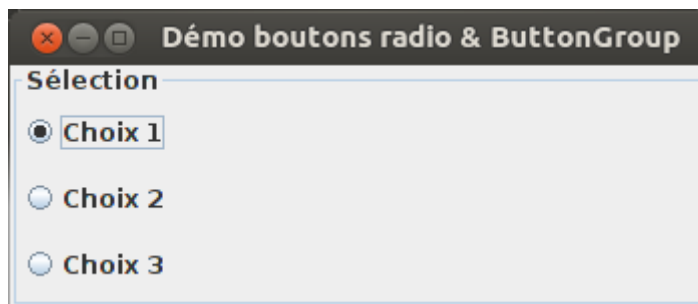
Présentation 1.13 Démonstration ButtonGroup et JRadioButton

```
1 import java.awt.*;
2 import javax.swing.*;
3 import javax.swing.border.Border;
4
5 public class DemoBoutonRadio extends JFrame {
```

```

6      public static void main(String args[]) {
7          DemoBoutonRadio app = new DemoBoutonRadio();
8          app.setVisible(true);
9      }
10     public DemoBoutonRadio() {
11         setTitle("D mo boutons radio & ButtonGroup");
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         setLocationRelativeTo(null);
14         JPanel panel = new JPanel(new GridLayout(0,1));
15         Border
16         BorderFactory.createTitledBorder("S lection");
17         panel.setBorder(border);
18         ButtonGroup group = new ButtonGroup();
19         JRadioButton radio1 = new JRadioButton("Choix 1", true);
20         JRadioButton radio2 = new JRadioButton("Choix 2");
21         JRadioButton radio3 = new JRadioButton("Choix 3");
22         group.add(radio1);
23         panel.add(radio1);
24         group.add(radio2);
25         panel.add(radio2);
26         group.add(radio3);
27         panel.add(radio3);
28         Container contentPane = this.getContentPane();
29         contentPane.add(panel, BorderLayout.CENTER);
30         setSize(300, 150);
31     }
32 }

```



14. La classe JCheckBox

Cette classe permet de cr er des cases   cocher.

On peut aussi regrouper plusieurs checkBox dans un JbuttonGroup.

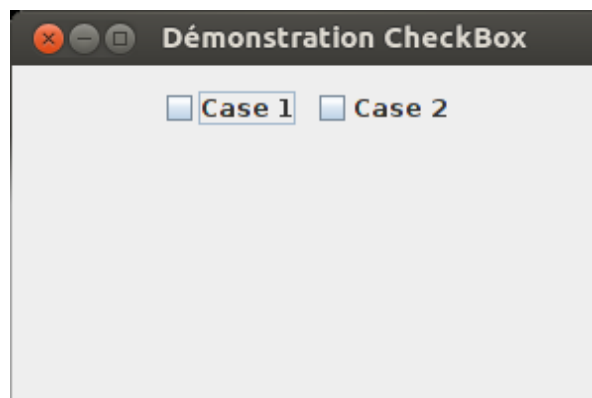
Pr sentation 1.14 D monstration de JCheckBox

```

1  import javax.swing.*;
2  import java.awt.*;
3
4  public class CheckBox extends JFrame{
5      JPanel container = new JPanel();
6      JCheckBox check1 = new JCheckBox("Case 1");
7      JCheckBox check2 = new JCheckBox("Case 2");

```

```
8
9     public CheckBox() {
10         setTitle("Démonstration CheckBox");
11         setSize(300, 200);
12         setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13         setLocationRelativeTo(null);
14         setLayout(new BorderLayout());
15         JPanel top = new JPanel();
16         top.add(check1);
17         top.add(check2);
18         container.add(top, BorderLayout.NORTH);
19         setContentPane(container);
20     }
21     public static void main(String[] args) {
22         CheckBox cb = new CheckBox();
23         cb.setVisible(true);
24     }
25 }
```



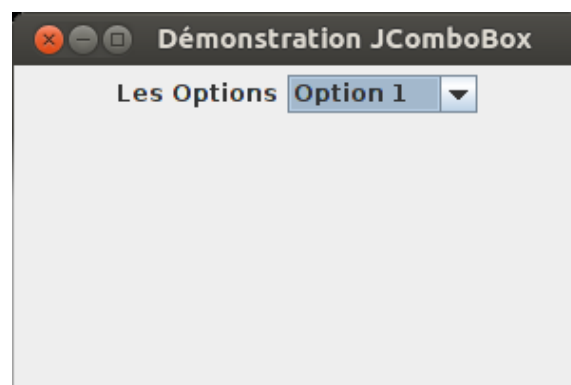
15. La classe JComboBox

Elle permet de créer une liste déroulante.

Lorsqu'on crée un objet JComboBox, on lui ajoute les éléments grâce à la méthode **addItem**.

Présentation 1.15 Démonstration JComboBox

```
1 import java.awt.*;
2 import javax.swing.*;
3
4 public class DemoComboBox extends JFrame {
5     JComboBox combo = new JComboBox();
6     JLabel label = new JLabel("Les Options");
7
8     public DemoComboBox() {
9         setTitle("Démonstration JComboBox");
10        setSize(300, 200);
11        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        setLocationRelativeTo(null);
13        setLayout(new BorderLayout());
14        combo.setPreferredSize(new Dimension(100, 20));
15        combo.addItem("Option 1");
16        combo.addItem("Option 2");
17        combo.addItem("Option 3");
18        combo.addItem("Option 4");
19        JPanel panel = new JPanel();
20        panel.add(label);
21        panel.add(combo);
22        add(panel, BorderLayout.NORTH);
23    }
24    public static void main(String[] args) {
25        DemoComboBox combobox = new DemoComboBox();
26        combobox.setVisible(true);
27    }
28 }
```



16. La classe JTable

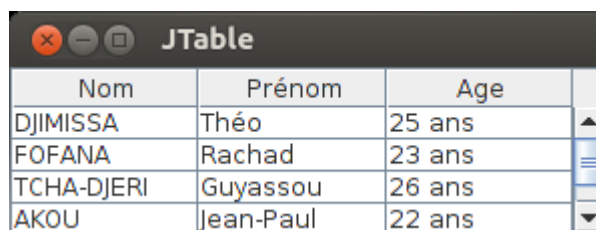
Cette classe permet de créer des tables. Les tables sont des composants qui permettent d'afficher les objets de façon structurée. Le constructeur de JTable prend deux paramètres le premier correspond à un tableau contenant des tableaux de String (String[][]), le second est un tableau de string (String[]) contenant les noms des colonnes de la table. (*Voir présentation ci-dessous*).

Présentation 1.16 Démonstration JTable

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 public class Fenetre extends JFrame {
5     public Fenetre(){
6         this.setLocationRelativeTo(null);
7         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
8         this.setTitle("JTable");
9         this.setSize(300, 120);
10        //Les données du tableau
11        String[][] data = {
12            {"DJIMISSA", "Théo", "25 ans"},
13            {"FOFANA", "Rachad", "23 ans"},
14            {"TCHA-DJERI", "Guyassou", "26 ans"},
15            {"AKOU", "Jean-Paul", "22 ans"},
16            {"ALABA", "Éric", "26 ans"}
17        };
18        //Les titres des colonnes
19        String[] title = {"Nom", "Prénom", "Age"};
20        JTable tableau = new JTable(data, title);
21        //Nous ajoutons notre tableau à notre contentPane dans
    un scroll
22        //Sinon les titres des colonnes ne s'afficheront pas !
23        this.getContentPane().add(new JScrollPane(tableau));
24    }
25    public static void main(String[] args){
26        Fenetre fen = new Fenetre();
27        fen.setVisible(true);
28    }
29 }

```



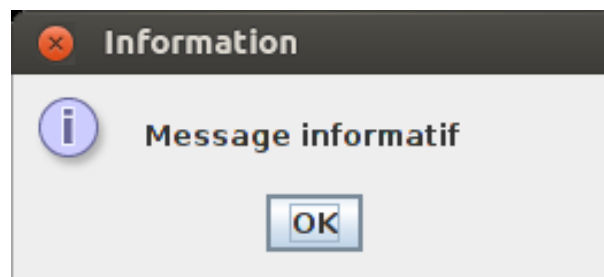
Nom	Prénom	Age
DJIMISSA	Théo	25 ans
FOFANA	Rachad	23 ans
TCHA-DJERI	Guyassou	26 ans
AKOU	Jean-Paul	22 ans

17. La classe JOptionPane

Cette classe permet de créer de petites fenêtres pouvant servir à afficher une information (un avertissement, un message d'erreur ...). Elle se trouve dans le paquetage javax.swing. Nous utiliserons la méthode **showMessageDialog** pour l'affichage de cette fenêtre.

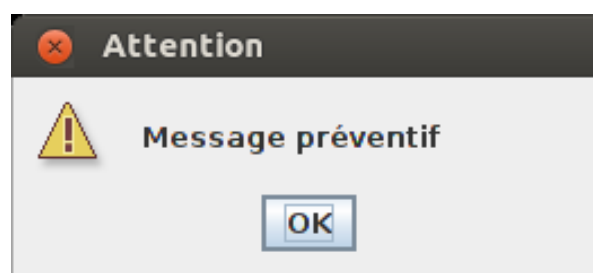
Présentation 1.17.1 Message informatif

```
1 import javax.swing.*;  
2  
3 public class OptionPane1{  
4     public static void main(String[] args) {  
5         JOptionPane jop1 = new JOptionPane();  
6         jop1.showMessageDialog(null, "Message informatif",  
7                               "Information", JOptionPane.INFORMATION_MESSAGE);  
8     }  
9 }
```



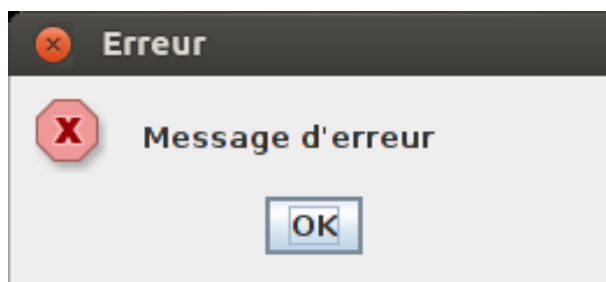
Présentation 1.17.2 Message préventif

```
1 import javax.swing.*;  
2  
3 public class OptionPane2{  
4     public static void main(String[] args) {  
5         JOptionPane jop2 = new JOptionPane();  
6         jop2.showMessageDialog(null, "Message  
7         préventif", "Attention", JOptionPane.WARNING_MESSAGE);  
8     }  
9 }
```



Présentation 1.17.3 Message d'erreur

```
1 import javax.swing.*;  
2  
3 public class OptionPane3{  
4     public static void main(String[] args) {  
5         JOptionPane jop3 = new JOptionPane();  
6         jop3.showMessageDialog(null, "Message  
d'erreur", "Erreur", JOptionPane.ERROR_MESSAGE);  
7     }  
8 }
```



La méthode **showMessageDialog** prend 4 paramètres

showMessageDialog (**Component** parentComponent, **String** message, **String** title, **int** messageType)

Component parentComponent : correspond au composant parent.

String message : permet de renseigner le message à afficher dans la boîte de dialogue.

String title : permet de donner un titre à l'objet.

int messageType : permet de savoir s'il s'agit d'un message d'information, de prévention ou d'erreur

II) Les événements

Un événement est une action sur l'interface qui est susceptible de déclencher une réaction.

Il faut noter que les événements représentent souvent des actions telles que cliquer la souris, appuyer sur une touche du clavier, cliquer sur le bouton de fermeture d'une fenêtre...

Dans la programmation événementielle Java (swing), les méthodes (**gestionnaires d'événements**) sont définies dans des interfaces appelés **listener**. Le processus est un peu similaire qu'en JavaScript ; il faut aussi abonner le listener au composant sur lequel se produira l'événement. Après cet abonnement du listener, le gestionnaire d'événement est automatiquement appelé quand l'événement survient avec l'événement comme argument (*Voir les présentations ci-dessous*).



1. Interagir avec le bouton : L'interface ActionListener

Vous aviez constaté que lorsqu'on clique sur les boutons que nous avons précédemment créés, rien ne se passe. Afin de gérer les différentes actions selon le bouton, nous allons utiliser l'interface ActionListener que nous implémenterons .

Par exemple, créons une fenêtre avec deux panels et deux bouton. Le premier panel se colorie en bleu lorsqu'on clique sur le bouton bleu et le second panel se colorie en vert lorsqu'on clique sur le bouton vert.

Présentation 2.1 Démonstration ActionListener

```
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4 import java.awt.*;
5
6 public class DemoActionListener extends JFrame implements
  ActionListener{
7     JPanel panelBleu = new JPanel();
8     JPanel panelVert = new JPanel();
9
10    public DemoActionListener(){
11        setTitle("Démonstration ActionListenerDemoFrame");
12        setSize(300, 200);
13        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14        setLocationRelativeTo(null);
15        setLayout(new BorderLayout());
16        JPanel panneau = new JPanel();
17        panneau.setLayout(new GridLayout(1, 2));
18        panneau.add(panelBleu);
```

```
19      panneau.add(panelVert);
20      JPanel panelBouton = new JPanel();
21      JButton bouton1 = new JButton("Bleu");
22      bouton1.addActionListener(this);
23      panelBouton.add(bouton1);
24      JButton bouton2 = new JButton("Vert");
25      bouton2.addActionListener(this);
26      panelBouton.add(bouton2);
27      add(panneau, BorderLayout.CENTER);
28      add(panelBouton, BorderLayout.SOUTH);
29
30  }
31  public void actionPerformed(ActionEvent e) {
32      if(e.getActionCommand().equals("Bleu")) {
33          panelBleu.setBackground(Color.blue);
34      }
35      else {
36          panelVert.setBackground(Color.green);
37      }
38  }
39  public static void main(String[] args) {
40      DemoActionListener action = new DemoActionListener();
41      action.setVisible(true);
42  }
43 }
```

Après un clique sur le bouton Bleu



Après un clique sur le bouton Vert



2. Interagir avec la souris : l'interface `MouseListener`

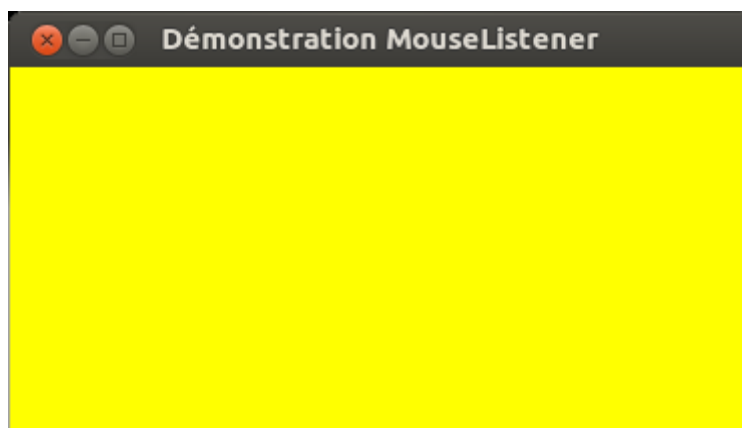
Créons une fenêtre qui se colore en Jaune lors du clique de souris.

Présentation 2.2 Démonstration `MouseListener`

```
1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseListener;
5
6 public class DemoMouseListener extends JFrame implements
    MouseListener{
7
8     public DemoMouseListener(){
9         setTitle("Démonstration MouseListener");
10        setSize(300, 200);
11        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        setLocationRelativeTo(null);
13        this.addMouseListener(this);
14    }
15    //Méthode appelée lors du clic de souris
16    public void mouseClicked(MouseEvent event) {
17        getContentPane().setBackground(Color.yellow);
18    }
19
20    //Méthode appelée lors du survol de la souris
21    public void mouseEntered(MouseEvent event) {}
22
23    //Méthode appelée lorsque la souris sort de la zone du bouton
24    public void mouseExited(MouseEvent event) {}
25
26    //Méthode appelée lorsque l'on presse le bouton gauche de la
    souris
27    public void mousePressed(MouseEvent event) {}
28
```

```
29 //Méthode appelée lorsque l'on relâche le clic de souris
30 public void mouseReleased(MouseEvent event) {}
31
32 public static void main(String[] args) {
33     DemoMouseListener mouse = new DemoMouseListener();
34     mouse.setVisible(true);
35 }
36
37 }
```

Après un clique sur la fenêtre, on a :



Rappelez-vous que même si vous n'utilisez pas toutes les méthodes d'une interface, vous devez malgré tout insérer le squelette des méthodes non utilisées (avec les accolades).

3. Contrôle du clavier : l'interface **KeyListener**

Cette interface nous permet d'intercepter les événements liés au clavier tels que : presser une touche, relâcher une touche, taper sur une touche.

Exemple : Créons une fenêtre qui se ferme lorsqu'on tape sur la touche **q**.

Présentation 2.3 Démonstration KeyListener

```

1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.KeyEvent;
4 import java.awt.event.KeyListener;
5
6 public class DemoKeyListener extends JFrame implements KeyListener{
7
8     public DemoKeyListener(){
9         setTitle("Démonstration KeyListener");
10        setSize(300, 200);
11        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12        setLocationRelativeTo(null);
13        this.addKeyListener(this);
14        setVisible(true);
15    }
16    //appelée lorsqu'on relâche une touche
17    public void keyReleased(KeyEvent event) {}
18
19    //appelée lorsqu'on presse une touche
20    public void keyPressed(KeyEvent event) {}
21
22    //appelée lorsqu'on tape sur une touche
23    public void keyTyped(KeyEvent event) {
24        if(event.getKeyChar()=='q'){
25            System.exit(0);
26        }
27    }
28    public static void main(String[] args) {
29        DemoKeyListener key = new DemoKeyListener();
30    }
31 }

```

4. L'interface WindowListener

Lorsque l'utilisateur clique sur le bouton de fermeture de la fenêtre (ou les deux autres boutons), le JFrame envoie un événement connu comme un événement de fenêtre (**window event**). La méthode utilisée pour enregistrer un window listener est *setWindowListener*. Les événements de fenêtre sont des objets de la classe **WindowEvent**.

Sept fonctions sont disponibles définies dans l'interface WindowListener :

Fonctions	Moment d'appel
public void windowOpened(WindowEvent e)	Appelée quand une fenêtre est ouverte.
public void windowClosing(WindowEvent e)	Appelée lorsqu'une fenêtre est en train de se fermer. Un clic sur le bouton de fermeture appelle cette fonction.
public void windowClosed(WindowEvent e)	Appelée lorsqu'une fenêtre se ferme.

	ferme.
<code>public void windowIconified(WindowEvent e)</code>	Invoked when a window is iconified. When you click the minimize button in a JFrame, it is iconified.
<code>public void windowDeiconified(WindowEvent e)</code>	Invoked when a window is deiconified. When you activate a minimized window, it is deiconified.
<code>public void windowActivated(WindowEvent e)</code>	Appelée lorsqu'une fenêtre est invoquée. Quand vous cliquez sur une fenêtre, elle devient la fenêtre activée.
<code>public void windowDeactivated(WindowEvent e)</code>	Invoked when a window is deactivated. When a window is activated, all other windows are deactivated. Other actions can also deactivate a window.

Présentation 2.4 Démonstration WindowListener

```

1 import java.awt.*;
2 import javax.swing.*;
3 import java.awt.event.*;
4
5 public class DemoWindowListener extends JFrame implements
   WindowListener{
6     public DemoWindowListener() {
7         setSize(400, 200);
8         setTitle("Demonstration Window Listener");
9         setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
10        setLocationRelativeTo(null);
11        addWindowListener(this);
12        getContentPane().setBackground(Color.LIGHT_GRAY);
13        JLabel aLabel = new JLabel(" Fenêtre avec un
   WindowListener");
14        add(aLabel);
15    }
16    public void windowOpened(WindowEvent e) {}
17
18    public void windowClosing(WindowEvent e) {
19        FenetreConfirmation confirm = new FenetreConfirmation();
20        confirm.setVisible( true);
21    }
22    public void windowClosed(WindowEvent e) {}
23
24    public void windowIconified(WindowEvent e) {}
25
26    public void windowDeiconified(WindowEvent e) {}
27
28    public void windowActivated(WindowEvent e) {}

```



```
29
30     public void windowDeactivated(WindowEvent e) {}
31
32     private class FenetreConfirmation extends JFrame implements
ActionListener {
33
34         public FenetreConfirmation(){
35             setSize(250, 100);
36             getContentPane().setBackground(Color.YELLOW);
37             setLayout( new BorderLayout());
38             setLocationRelativeTo(null);
39             JLabel confirmLabel = new JLabel("Etes-vous sûr de
vouloir quitter ?");
40             add(confirmLabel, BorderLayout.CENTER);
41
42             JPanel buttonPanel = new JPanel();
43             buttonPanel.setBackground(Color.ORANGE);
44             buttonPanel.setLayout( new FlowLayout());
45
46             JButton exitButton = new JButton("Oui");
47             exitButton.addActionListener( this);
48             buttonPanel.add(exitButton);
49
50             JButton cancelButton = new JButton("Non");
51             cancelButton.addActionListener( this);
52             buttonPanel.add(cancelButton);
53             add(buttonPanel, BorderLayout.SOUTH);
54         }
55         public void actionPerformed(ActionEvent e){
56             String actionCommand = e.getActionCommand();
57             if (actionCommand.equals("Oui"))
58                 System.exit(0);
59             else {
60                 dispose(); }
61         }
62     }
63     public static void main(String[] args) {
64         DemoWindowListener listener = new DemoWindowListener();
65         listener.setVisible(true);
66     }
67 }
```

